

# Arquitetura de Sistemas & Computadores II

Técnica: Vasco Pedro

vp@di.menora.pt

http://www.di.menora.pt/~vp

Morânia Atendimento - 3as: 14h30 - 19h30

Gabinete 256

Prática - Pedro Patho

pp@di.menora.pt

http://www.di.menora.pt/~pp

Gabinete 256

1a Freqüência - 29 de Outubro: 18h30

2a Freqüência - 17 de Dezembro: 18h30

Exame Época Normal - 8 de Janeiro: 14h30

Exame de Recurso - 22 de Janeiro: 14h30

Livros - Computer Organization

See Mips Run

Programa - Análise de Desempenho

- Implementações de Micromecanismos
  - Controle e caminho de dados
  - Implementação microciclo
    - Pipeline
  - Organização da Memória
    - Memória Cache
    - Análise de desempenho
  - Sistemas de Armazenamento
    - Organizações e desempenho
    - I/O
    - Paralelismo e Multiprocessamento

(Prática)

1-

- a) calcula o número de retas  
b)

lw v0, 0(a0)

ciclo: addi a1, a1, -1

beq a1, zero, fim

addiu a0, a0, 4

lw t0, 0(a0)

sub t1, t0, t0

beq t1, zero, ciclo

or v0, t0, t0

j ciclo

nop

fim: j a. no.

nop

2-

a) a0  $\rightarrow$  endereço A

a1  $\rightarrow$  endereço B

a2  $\rightarrow$  0 \$NS 45000

v0  $\rightarrow$  resultado

lw t0, 0(a0)

lw t1, 0(a1)

mul t2, a2, a2

ciclo: addi t3, t1, -1

beq t1, zero, fim

nop

add t3, t0, t1

sw t3, 0(v0)

addiu a0, a0, 4

addiu a1, a1, 4

addiu v0, v0, 3

lw t0, 0(a0)

lw t1, 0(a1)

j ciclo

nop

4-

$a0 \rightarrow$  endereço do vetor  
 $a1 \rightarrow$  dimensão do vetor ( $> 0$ )  
 $a2 \rightarrow x$

lw t0, 0(a0)

or v0, zero, zero

add a1, a1, -1

ciclo: beg, a1, zero, fin

nop

seq t1, t0, a2

add v0, v0, t1

add a0, a0, 1

lw t0, 0(a0)

j ciclo

nop

fin: jr ra

nop

seq t1, t0, a2

add v0, v0, t1

add v1, v1, t2

ciclo: beg a1, zero, fin

nop

seq t1, t0, a2

add v0, v0, t1

add a0, a0, 1

lw t0, 0(a0)

j ciclo

nop

fin: jr ra

nop

seq t1, t0, a2

add v0, v0, t1

add a0, a0, 1

lw t0, 0(a0)

j ciclo

nop

fin: jr ra

nop

5-

$a0 \rightarrow$  endereço do vetor  
 $a1 \rightarrow$  dimensão do vetor  
 $a2 \rightarrow x$

lw t0, 0(a0)

or v0, zero, zero

add a1, a1, -1

ciclo: beg a1, zero, fin

nop

seq t1, t0, a2

add v0, v0, t1

add v1, v1, t2

add v2, v2, t3

add v3, v3, t4

lw t0, 0(a0)

j ciclo

nop

add a1, a1, -1

fin: jr ra

nop

add a1, a1, -1

lw t0, 0(a0)

j ciclo

nop

add a1, a1, -1

lw t0, 0(a0)

j ciclo

nop

add a1, a1, -1

lw t0, 0(a0)

j ciclo

nop

add a1, a1, -1

lw t0, 0(a0)

j ciclo

(Técnica)

Menciono Aprendizado -

frequências (Material) - folha A4

- Calendárcio simples

Handwriting e Software

Aplicações  
Software Sistemas

Handwriting ← Arquitetura de conjunto de instruções (Interface)

## Execução de um programa

- 1) Programa está em disco, numa memória, em alguma rede
  - 2) Programa é carregado para a memória do computador
  - 3) Instruções são executadas pelo processador
    - Execuções são lidas da memória e executadas pelo processador
    - Dados são lidos da memória e escritos na memória
- ↳ resultado é escrito em disco ou no screen

## Análise do desempenho

Como medir o desempenho de um computador?

- Tempo de execução de um programa (computador pessoal)
- Número de instruções realizadas por unidade de tempo (throughput) (server)

## Comparação do desempenho

Quanto menor for o tempo de execução de um programa, maior será o desempenho

$$\text{Desempenho} \propto \frac{1}{\text{tempo de execução}}$$

$$\Rightarrow \text{Desempenho}_x > \text{Desempenho}_y \Rightarrow \frac{1}{\text{tempo execução}_x} > \frac{1}{\text{tempo execução}_y} \Rightarrow \\ \Rightarrow \text{tempo execução}_x < \text{tempo execução}_y$$

• Não há medida para medir o desempenho

## Comparação quantitativa

Se  $n > 1$ , o computador  $x$  é  $n$  vezes mais rápido que o  $y$

$$\frac{\text{Desempenho}_x}{\text{Desempenho}_y} = \frac{\text{tempo de execução}_y}{\text{tempo de execução}_x} = n$$

## Tempo de execução

- tempo decorrido entre o inicio e o fim da execução do programa (wall clock time, response time ou elapsed time)

→ pode ser afetado pela larga da máquina  
→ apropriado para programar paralelos

- Tempo de CPU, o tempo durante o qual o CPU estiver a trabalhar para o programa (CPU time) que se divide em:
  - tempo a executar instruções do programa (User CPU time)
  - tempo a executar instruções do sistema operativo em prol do programa (System CPU time)

### Relógio

- Processador é regido através de um sinal de relógio
- Período - tempo de duração de um ciclo
- Frequência (clock rate) - inverso do período

Tempo de CPU = nº de ciclos x duração de 1 ciclo

ou

$$\text{Tempo de CPU} = \frac{\text{nº de ciclos}}{\text{frequência do relógio}}$$

Ovenclock → Psn o processador a uma velocidade de relógio rápida

• CPI (clock cycles per instruction) → número médio de ciclos psn instruções

$$\text{nº de ciclos} = \text{nº de instruções} \times \text{CPI}$$

- Diferentes instruções podem necessitar de diferentes números de ciclos
- O CPI pode ser utilizado para comparar diferentes implementações da mesma arquitetura

(Prática)

1.5

$$P_1 \rightarrow 3\text{GHz}, \text{ CPI } 1,5$$

$$P_2 \rightarrow 2,5\text{GHz}, \text{ CPI } 1$$

$$P_3 \rightarrow 1,6\text{GHz}, \text{ CPI } 2,2$$

a) Melhor desempenho → P2

$$\frac{1,5}{P_1} \cdot \frac{1}{P_2} \cdot \frac{2,2}{P_3} \longrightarrow \frac{1,5}{3 \times 10^9} > \frac{1}{2,5 \times 10^9} < \frac{1,5}{1,6 \times 10^9}$$

$$\downarrow \quad \downarrow \quad \downarrow \\ 0,5 \quad 0,4 \quad 0,55$$

b) 10 segundos, numero ciclos?

$$CPI = \frac{n^{\circ} \text{ ciclos}}{n^{\circ} \text{ instruções}}$$

$$\rho_1 - n^{\circ} \text{ de ciclos} = 10 \times 3 \times 10^9 = 30 \times 10^9 = 30 \text{ mil milhões de ciclos}$$

$$n^{\circ} \text{ instruções} = \frac{30 \times 10^9}{4,5} = 20 \times 10^9$$

$$\rho_2 - n^{\circ} \text{ de ciclos} = 10 \times 2,5 \times 10^9 = 25 \times 10^9 = 25 \text{ mil milhões de ciclos}$$

$$n^{\circ} \text{ instruções} = \frac{25 \times 10^9}{7,2} = 25 \times 10^9$$

$$\rho_3 - n^{\circ} \text{ de ciclos} = 10 \times 4 \times 10^9 = 40 \times 10^9 = 40 \text{ mil milhões de ciclos}$$

$$n^{\circ} \text{ instruções} = \frac{40 \times 10^9}{7,2} = 19 \times 10^9$$

c) + 20% CPI aumento

- 30% tempo de execução

$$T = 10 \rightarrow 7 (-30\%)$$

	CPI + 20%	nº ciclos total	freq. total
	clock	A B C D	
$\rho_1$	3,5 GHz	1 2 3 3	$\frac{1 \times 10^5 + 2 \times 2 \times 10^5 + 3 \times 3 \times 10^5 + 3 \times 2 \times 10^5}{10^6} = \frac{26 \times 10^5}{10^6} = 2,6$
$\rho_2$	3 GHz	2 2 2 2	2

7.6

	CPI		
	clock	A B C D	
$\rho_1$	1,8	36 $\times 10^9$	5,11 GHz
$\rho_2$	1,2	$30 \times 10^9$	3,29 GHz

$$\rho_3 = 2,61 \quad 4,2 \times 10^9$$

$$6,79 \text{ GHz}$$

	CPI		
	clock	A B C D	
A	10 <sup>5</sup>		
B	$2 \times 10^5$	$\rho_1$	$3,6 \times 10^6 / 2,5 \times 10^4$
C	$5 \times 10^5$	$\rho_2$	$2 \times 10^6 / 3 \times 10^9$
D	$2 \times 10^5$		

$$\text{total ciclos}$$

$$3,6 \times 10^6 / 2,5 \times 10^4$$

$$2 \times 10^6 / 3 \times 10^9$$

(Técnica)

CPI de um conjunto de instruções

Uma arquitetura inclui vários tipos de instruções

- Aritméticas
- Acesso à memória
- Saltos (condicionais ou não)

As diferentes classes podem apresentar CPIs diferentes

$$CPI_{Global} = \sum_{i=1}^{N_{classes}} \%_i \cdot CPI_i = 60\% \cdot 1 + 30\% \cdot 3 + 10\% \cdot 7 = 7,7$$

Equação clássica do desempenho do CPU

Tempo de CPU = n de instruções x (CPI x Duração de 1 ciclo)

$$\text{Tempo de CPU} = \frac{n_{instruções} \times CPI}{frequência \text{ relógio}}$$

Fatores de desempenho

O que é medido

Tempo de CPU

Segundos a executar o programa

Número de instruções

Instruções executadas para o programa

CPI

Número médio de ciclos por instrução

Duração de 1 ciclo /

Segundos por ciclo de relógio

frequência do relógio

Ciclos de relógio por segundo

Lei de Amdahl

$$\text{Tempo depois da melhoria} = \frac{\text{tempo afetado pela melhoria}}{\text{Veloc. da melhoria}} + \frac{\text{tempo não afetado}}{\text{pela melhoria}}$$

Amdahl → pensam que a melhoria de um aspecto de um computador levaria a um aumento proporcional do desempenho global

MIPS → milhões de instruções por segundo; mais rápido a executar instruções

$$MIPS = \frac{n_{instruções}}{\text{tempo de execução}} \times 10^6 = \text{frequência} \cdot CPI$$

Com a mesma frequência executaria mais de pressa o que tiver menor CPI

Ano-dilhado → Usam só parte da equação de desempenho para caracterizar o desempenho

### Problemas:

- 1) Não é possível usar para comparar computadores com diferentes conjuntos de instruções
- 2) O valor de MIPS para um computador depende do programa usado para o cálculo
- 3) Se uma nova versão de um programa executa mais instruções e cada instrução é mais rápida, o valor de MIPS pode aumentar mesmo que o desempenho diminua

### Implementação de microprocessadores

Um processador consiste em:

- Caminho de dados (data path)
- Controle

### Instruções MIPS

- Aritméticas - add, sub, and, or, sllt
- Acesso à memória - lw, sw
- Salto condicional - beq
- Salto incondicional - j

### Execução de uma instrução

Fazem do ciclo de execução de uma instrução máquina pelo processador:

Fetch - Leitura / Transferência da instrução para o processador

Decode - Descodificação / Identificação da instrução

Execute - Execução da instrução (Depende da instrução, pode ser mais simples ou mais complexa)

### Execução num processador MIPS

Fetch - 1) Leitura da instrução no endereço da memória de instruções contido no registo PC (program counter)

- 2) Cálculo do endereço da instrução seguinte (PC + 4)

Execute - Memória de instruções, PC e somador

Instrução add

instrução do tipo R

Execução:

- 1) leitura da instrução
- 2) leitura do conteúdo dos registros rs e rt
- 3) cálculo da soma
- 4) Escreva do resultado no registro rd

Envolve - Banco de registros & ALU

(Regwrite - controle para verificar se o registro vai ser escrito ou não)

Operações da ALU (unidade aritmética e lógica)

0000	→ and
0001	→ or
0010	→ add
0110	→ sub
0111	→ slt
1100	→ nor

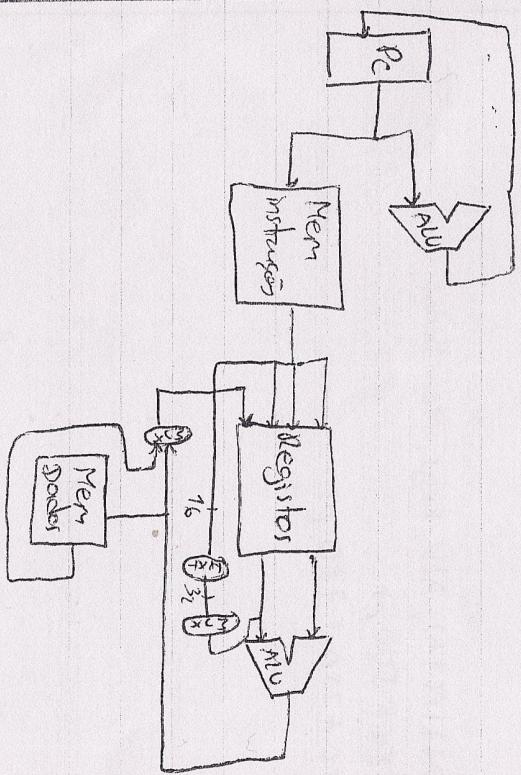
Instruções lw

instrução do tipo I

Execução:

- 1) leitura da instrução
- 2) leitura do conteúdo dos registros rs
- 3) cálculo do endereço de memória a escrever: im + rs
- 4) leitura do valor presente no endereço calculado
- 5) Escreva do valor lido no registro rt

Envolve - Memória de dados & extensão de sinal



Ext - Extensão de Sinal

MUX - Multiplexer

(Prática)

4.1

4.1.1	Instruction → and rd, rs, rt	tipo R				
RegWrite	RegDst	MemWrite	Mem Read	ALUsrc	ALUOp	MemToReg

and

4.1.2 Registros, ALU

4.1.3 1ª parte -

1ª parte - Sign-extend, Memória Dados

4.1.X

RegWrite	RegDst	MemWrite	MemRead	ALUsrc	ALUOp	MemToReg
lw	1	0	0	1	1	0
sw	0	x	1	0	1	0

4.1.2 lw e sw - Data Memory, Sign-extend

4.1.3 Usa todos

4.1.4 Operações realizada pelo ALU

And - x

lw - soma

sw - soma

4.2

4.2.1 Registros, Memória Dados, ALU

4.2.2 Nenhuma

4.2.3 ALUsrc, MemToReg, RegWrite, MemRead

A3 -

a) 3x Mux, Mem instâncias, Add1PC + 4, ALU, Regs, Mem Dados

$$3 \times 20 + 200 + 70 + 90 + 80 = 500 \text{ ps}$$

$$b) 3 \times 20 + 200 + 70 + 90 + 80 + 250 = 750 \text{ ps}$$

### (Teórica)

Os registos são sempre lidos sejam usados ou não

Siw #9, zero (\$8) usa 2 registos  
lw \$9, zero (\$8) usa o valor de 1 registo

### Instrução beg

#### Instrução do tipo I

- Operações:
- 1) Leitura da instrução
  - 2) Leitura do conteúdo dos registos rs e rt
  - 3) Comparação
  - 4) Escrita do endereço da próxima instrução a executar

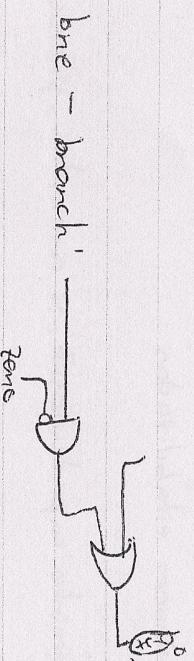
$$PC + imm \times 4 + 4$$

Endereçamento - Registros, ALU e Soma/dec

Usa 2 vezes a ALU e faz a diferença entre 2 registos

$$\overline{AUCP_1 AND F_2} = \overline{ALUOP_1} \text{ on } \overline{F_2}$$

- RegDst - escolhe o registo que vai ser lido
- MemtoReg - escolhe o valor que vai ser escrito no registo



- A memória de dados não tem saída
- Em cada momento todos os endereços estão desligados menos 1

### Instrução jump

#### Instrução do tipo J

Não usa registos nenhum

Todas as instruções estão alinhadas aos 32 bits / 4 bytes  
Todos os endereços são múltiplos de 4

Estratégia: 1) Leitura da instrução

2) Cálculo do endereço da próxima instrução a ser executada

$  PC + 4  $	imm	$  0   0  $
$3_1$	$2_8$	$2_0$

Caminho de dados completo

- Implementação monociclo - instrução consegue na parte ascendente do relógio ( $CPI = 1$ )
- Todas as instruções executam suas tarefas no inicio ciclo de relógio
  - Não há delay slot

Fases de execução de add, lw e beq

add rd, rs, rt	lw rt, imm(rs)	beq rs, rt, imm
Leitura de rs e rt	Leitura da instrução	Leitura de rs e rt
Soma de rs e rt	Soma de rs e imm	Comparação de rs e rt
Escrita de rd	Acervo à memória	Escrita de rt

Duração de um ciclo

Latença de um circuito - Tempo desde que os valores estão presentes nos entradas do circuito até as suas saídas estarem estabilizadas

Caminho crítico de uma instrução - Caminho cuja duração, se aumentada, provoca o aumento da duração da instrução;

Compreende a sequência de operações efetuadas durante a execução da instrução cuja duração é mais longa

Duração de ciclo de relógio tem de acomodar a instrução cujo caminho crítico é mais longo

A instrução mais longa é lw  $\rightarrow T \geq 800\text{ps}$

- ALU só é usada para calcular valores em endereços, por isso só é usado 1 vez

### Execução de instruções

- sequencial
- em paralelo

### Execução pipeline

- Execução em paralelo
- Não se usam implementações monociclo nem sim pipeline

$$\text{Tempo entre instruções pipeline} = \frac{\text{Tempo entre instruções não pipeline}}{\text{Número de andares do pipeline}}$$

- Duração do ciclo de relógio diminui
- Tempo para executar uma instrução não diminui
- Faz de mesmo arranjo, pois se os andares do pipeline não são perpendicularmente equilibrados
- Todos os instruções levam o mesmo tempo
- Aumenta o número de instruções executadas por unidade de tempo (throughput)

### Arquitetura MIPS e pipelines

- Todas as instruções têm o mesmo tempo de execução e podem ser lidas no primeiro ciclo no pipeline e descodificadas no segundo
- Pocos formatos diferentes de instruções e com os registos nas mesmas posições, pelo que é possível iniciar a sua leitura em simultâneo com a descodificação
- Só loads e stores lidam com endereços e não é necessário usar a ALU para cálculo de endereços e para outra operação na mesma instrução
- Valores armazenados em memória permitem que um único acesso seja suficiente para os transferir
- Instruções só produzem um valor que é escrito no último andar do pipeline

- Cada fase corresponde a um andar do pipeline

Andar	Função	Unidade funcional principal
IF	Instruction fetch	Memória de instruções
ID	Instruction decode	Banco de registos (leitura)
Ex	Execute	ALU
MEM	Memory access	Memória de dados
WB	Write Back	Banco de registos (escrita)

Problemas inerentes aos pipelines

- Conflitos estruturais (structural hazards)

Quando não é possível executar uma combinação de instruções no mesmo ciclo  
É a razão para o MIPS ter regras de incompatibilidade de instruções e de dados

- Conflitos de dados (data hazards)

Quando uma instrução necessita de um valor produzido por uma instrução anterior e ainda não disponível

- Conflitos de controlo (control or branch hazards)

É necessário saber o destino de um salto condicional para poder executar a próxima instrução

- Que fazer quando se encontra um salto condicional?

- 1) Ativam o pipeline até saber que instrução deverá ser executada;
- 2) Assumir que o salto não será efectuado e começam a executar a instrução que se segue

3) Assumir que os saltos condicionais para instruções anteriores (como os que existem no fim de um ciclo) serão efectuados

4) Empregam técnicas mais sofisticadas para tentar prever se um salto será ou não efectuado

• Quando a previsão se revelar errada, o pipeline deve ser "limpo" das instruções que não deveriam ter sido executadas

• O MIPS usa delayed branches para minimizar os efeitos adversos dos conflitos de controlo

Atrazo de pipeline - O inicio da execução da instrução seguinte é atrasado um ciclo de reloj (latency) que a deixa sobre o salto a efectuado e torna

no andam ID do pipeline)

- A execução dos saltos condicionais pega a deroram 6 ciclos (vs 5 ciclos para os restantes instruções)

→ Os registros do pipeline guardam a informação necessária sobre a instrução a executar em cada andam: instrução, conteúdo dos registos, resultado da ALU, valor lido da memória, ... (lw)

IF - (unidades funcionais ativas) mux(PCSne), PC, somador, memória de instruções (Leitura), registo IF / ID (leitura)

ID - (unidades funcionais ativas) registo IF / ID (Leitura), banco de registos (Leitura) extensão de sinal, registo ID / EX (escrita)

EX - (unidades funcionais ativas) registo ID / EX (Leitura), mux(ALUSne), ALU, registo EX / MEM (escrita)

MEM - (unidades funcionais ativas) registo EX / MEM (Leitura), memória de dados (Leitura), registo MEM / WB (escrita)

WB - (unidades funcionais ativas) registo MEM / WB (Leitura), mux(MemtoReg), banco de registos (escrita)

(Prática)

i	8	IF	ID	EX	MEM	WB
		250 ps	350 ps	150 ps	300 ps	200 ps

$$t_{8.1} \text{ Cicle de relógio pipelined} = 350 \text{ ps}$$

$$\text{Cicle de relógio não pipelined} = 250 + 350 + 150 + 300 + 200 = 1250 \text{ ps}$$

$$t_{8.2} \text{ Latência pipeline} = 5 \times 350 \text{ ps} = 1750 \text{ ps}$$

$$\text{Latência não pipeline} = 250 + 350 + 150 + 300 + 200 = 1250 \text{ ps}$$

Latência → ps

$$ps = 10^{-12}, \text{ CPI pipelined} = 1$$

4.8.3 Partir um ordenado pelo - ID  $350 \rightarrow 175\text{ps} + 175\text{ps}$   
dividindo de maior latência

$$t = \frac{n_{ciclos}}{f} = \frac{n_{instruções} \times CPI}{f} = n_{instruções} \times CPI \propto T$$

CPI: - 1 instrução - 5 ciclos

2 " " - 5 + 1 ciclos

3 " " - 6 + 1 ciclos

4 " " - 7 + 1 ciclos

-  $10^6$  instruções -  $4 + 10^6$  ciclos

$10^6$ , 1 encadeada 5 e others 1 ciclo

$$CPI = \frac{\sum 10^6 + 5}{10^6}$$

ALU	beg	lw	sw
%	45	20	15

4.8.4 A memória é usada em 35% dos ciclos (sw + lw)

4.8.5 O write register é usado em 65% dos ciclos (alu + lw)

$$4.8.6 \text{ Speedup} = \frac{\text{Desempenho}_6}{\text{Desempenho}_5} = \frac{\text{Tempo}_5}{\text{Tempo}_6} = \frac{n_{instr} \times CPI \times T_5}{n_{instr} \times CPI \times T_6} = \frac{T_5}{T_6} = \frac{350}{300} = 1.17$$

t6 andares é 1.17 mais rápido que t5 andares

4.9

1 2 3 4 5 6 7 8 9 10 11

on \$1, \$2, \$3	IF ID EX MEM WB
on \$2, \$1, \$4	IF ID (ID) (ID) EX MEM WB
on \$1, \$1, \$2	IF (ID) EP ID (ID) (ID) EX MEM WB

4.9.12 On \$1, \$2, \$3

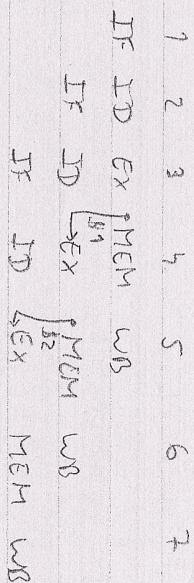
$\overrightarrow{\text{rotina}}$  } nops

On \$2, \$1, \$4 } nops

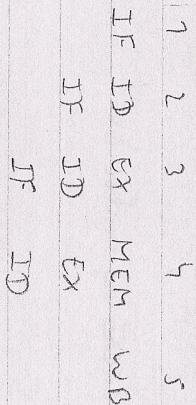
On \$1, \$2, \$3 } nops

4.9.3 full-forwarding (sempre que o valor está disponível) rode seu uso de onde

(já mencionado)



#### 4.9.6 Só MEM-EX (com ALU - ALU)



(Teórica)

(SW)

Ex - Unidades funcionais outras) registo ID / EX (leitura), mux (ALU/Sinc), ALU, registo EX / MEM (escrita, inclui rt)

MEM - (unidades funcionais outras) registo EX / MEM (leitura), memória de dados (escrita)

WB - não tem unidade funcional outras

• O registo a escrever é o da instrução que está no andor WB

(controle no pipeline)

- Correça no andor ID
- Os registos das andares do pipeline também guardam os mais de controlo para a instrução a executar

Dependência ≠ conflitos

↳ quando utiliza o valor de uma instrução anterior

Supõe a defesa de conflitos

- O registo ID / EX devem identificar os registos rs, rt, rd
- Os registos EX / MEM e MEM / WB devem identificar o registo rd

## Detecção de conflitos

Ou não pode ser usado porque só produz o valor no MEM

Ex	MEM	WB
add	lw	l
	\$7	\$7

(De MEM para Ex)

- Se houverá conflito se a instrução (tipo-R) que esteja no andar MEM
  - Escrever um registo
  - esse registo é o resultado da instrução no andar EX
  - esse registo não é \$0

(De WB para Ex)

- Só haverá conflito se a instrução que está no andar WB
  - escrever um registo
  - esse registo é o resultado da instrução no andar EX
  - esse registo não é \$0

## Multiplos Conflitos

add	\$1	\$1	\$2
add	\$1	\$1	\$3
add	\$1	\$1	\$4

Introdução de um ônibus no pipeline

Através do and um ciclo de relogio consiste em

- 1) Impedi-lo de avançar no pipeline - O novo sinal IF/ID manteve ter valor 0 pena que o registo IF/ID mantenha a instrução and;

- 2) Impedir uma nova instrução de entrar no pipeline - O novo sinal PCWrite tem valor 0 para que o PC mantecha o endereço de vez

- 3) Inserir um漏p entre o lw e add - É selecionada a entrada com valor 0 do novo multiplexor para os símbolos de controlo no registo IF/EX, de modo a estes tenham o valor zero, mantendo uma 'bolha' no andar EX do pipeline, entre o lw e o add.

lru \$2, 20(\$1)

and \$4, \$2, \$5

or \$8, \$2, \$6

add \$9, \$4, \$2

slt \$1, \$6, \$7

### Os saltos condicionais & o pipeline

- O endereço da instrução a executar a seguir a uma instrução de salto condicioinal só é conhecido depois de o processador ter tido oportunidade de ver os valores dos registos e de os comparar.

- O processador vai continuar a introduzir instruções no pipeline, antes de saber o endereço da instrução que deverá ser executada, para não desperdiçar ciclos de relógio.

- Para minimizar o trabalho feito em vós quando as instruções executadas não são as que devem ser, o processador tenta adivinhar (ou prever) se o salto será efectuado.

- Se a previsão estiver errada, parte do trabalho do processador foi inválida e o pipeline tem de ser limpo.

- Quanto mais precisa for a previsão, menos serão os ciclos desperdiçados em trabalho inútil e melhor será o desempenho.

### Prenvisão do comportamento de um salto condicional

Estratégias usadas para prever o efeito de um salto condicional

- Fixa - O salto não será efectuado. O processador continua a executar as instruções sequencialmente.

- Ciclos - Saltos de volta ao início do ciclo serão efectuados. Se o salto for para trás (para o inicio do ciclo), o processador vai executar as instruções a partir do endereço para onde o salto é efectuado, caso contrário continua sequencialmente.

Pensado - O salto terá o mesmo comportamento que da vez anterior. O processador prevê que a instrução terá o mesmo efeito que teve na última vez que foi executada. São ignorados o endereço e o destino dos saltos. Entra duas vezes em cada ciclo.

Histórica - Usa a história do comportamento da instrução. O efeito da instrução é previsto recorrendo ao seu comportamento pensado.

Global (complementing predictor) - Escolhe estratégia baseada em outros saltos. Pode usar uma previsão histórica se o salto anterior foi efectuado e outra se não foi.

Torneio (tournament branch predictor) - Encalha estratégia com maior sucesso. Recorre à estratégia que, até ao momento, se revelou mais precisa.

### Conflito de Controlo

- Quem executará a instrução a seguir não sabe qual é

Ex: beq

Saltos condicionais no MIPS

- O MIPS reduz o despendido de ciclos de relogio
- 1) Decidindo no orden ID do pipeline se efectua ou não o salto
- 2) Usando um delay slot (executa sempre a instrução a seguir)

Parecendo a decisão de beq para o orden ID, haverá no máximo uma espera no pipeline quando é decidido se o salto será efectuado

- Se o delay slot, se o salto for efectuado, a instrução incorrecta será substituída por um NOP

### Exceções

- Uma exceção envolve uma circunstância excepcional, ocorrida durante o processamento, que requer atençao com urgência
- Uma exceção pode ter origem interna ou externa ao processador
- As exceções externas ao processador são também conhecidas como interrupções.